Accelerating TensorFlow with Adaptive RDMA-based gRPC

Rajarshi Biswas, Xiaoyi Lu, Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering
The Ohio State University
{biswas.91, lu.932, panda.2}@osu.edu

***Abstract*—Google's TensorFlow is one of the most popular Deep Learning frameworks nowadays. Distributed TensorFlow supports various channels to efficiently transfer tensors, such as gRPC over TCP/IP, gRPC+Verbs, and gRPC+MPI. At present, the community lacks a thorough characterization of distributed TensorFlow communication channels. This is critical because high-performance Deep Learning with TensorFlow needs an efficient communication runtime. Thus, we conduct a thorough analysis of the communication characteristics of distributed TensorFlow. Our studies show that none of the existing channels in TensorFlow can support adaptive and efficient communication for Deep Learning workloads with different message sizes. Moreover, the community needs to maintain these different channels while the users are also expected to tune these channels to get the desired performance. Therefore, this paper proposes a unified approach to have a single gRPC runtime (i.e., AR-gRPC) in TensorFlow with Adaptive and efficient RDMA protocols. In AR-gRPC, we propose designs such as hybrid communication protocols, message pipelining and coalescing, zero-copy transmission etc. to make our runtime be adaptive to different message sizes for Deep Learning workloads. Our performance evaluations show that AR-gRPC can significantly speedup gRPC performance by up to 4.1x and 2.3x compared to the default gRPC design on IPoIB and another RDMA-based gRPC design in the community. Comet supercomputer shows that AR-gRPC design can reduce the Point-to-Point latency by up to 75% compared to the default gRPC design. By integrating our AR-gRPC with TensorFlow, we can achieve up to 3x distributed training speedup over default gRPC-IPoIB based TensorFlow.**

I. Introduction

Deep Learning (DL) is one of the fastest-growing field in Artificial Intelligence (AI). Google's TensorFlow is one of the most popular frameworks to perform distributed Deep Learning on big data sets and in the last few years it has gained a lot of momentum in Big Data, Deep Learning, and High-Performance Computing (HPC) communities.

During DL model training and inference on TensorFlow, gradient updates (or tensor transmissions) are the critical time-consuming steps that incur a massive volume of data transfer over the network. This becomes a major bottleneck in DL workloads. Increasing the mini-batch size is one solution as this results in less gradient updates and longer local computation in TensorFlow. However, studies [1, 2, 3] have shown this approach can increase the time for the DL model to converge. Other alternative solutions have been proposed to accelerate TensorFlow by taking advantage of various high-performance technologies. For instance, the current open-source TensorFlow leverages multiple ways of doing gradient updates, by running default gRPC [4] over TCP/IP or IPoIB (IP over InfiniBand), gRPC with a dedicated Verbs-based channel [5], and gRPC with a dedicated MPI [6, 5] based channel. The main reason of bringing Verbs and MPI based channels into TensorFlow is to utilize high-performance communication mechanisms such as Remote Direct Memory Access (RDMA) over high-speed interconnects, like InfiniBand and RDMA over Converged Ethernet (RoCE).

This research is supported in part by National Science Foundation grants #CNS-1513120, #IIS-1636846, and #CCF-1822987. It used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

According to recent studies [7, 8, 9, 10, 11] in the community, researchers have shown that with the help of high-performance interconnects and high-speed communication protocols, like RDMA, many system software such as Apache Hadoop, Spark, Key-Value Stores, and even Deep Learning frameworks can be benefited by incorporating native RDMA support in these frameworks.

A. Motivation

As we can see, in order to achieve the optimal communication performance on high-performance networks, the TensorFlow community has maintained different channels, i.e. gRPC, Verbs, and MPI. On the other hand, the users also need to understand and tune these channels on their platforms to get the desired performance. Such scenarios bring a lot of challenges for both developers as well as end users. This also motivates us to answer a broad question: Can a unified approach be proposed to provide optimal performance for TensorFlow workloads? To answer this question, we first conduct a survey on the existing solutions which have been proposed in the community for TensorFlow. Table I summarizes the comparison among these different solutions. From

Table I: Comparison with Related Work

Work	Channel	Main Mechanism
[12]	gRPC	TCP/IP, IP-over-IB
	gRPC + Verbs	RDMA for Tensor transfers; gRPC for others
	gRPC + MPI	MPI for Tensor transfers; gRPC for others
[13]	gRPC	Replacing Sockets Send/ Recv with Verbs Send/Recv
This paper	AR-gRPC	Native RDMA; Adaptive communication for TensorFlow demands

this table, we clearly see that the community is trying to run DL workloads on top of gRPC (with TCP/IP or IPoIB), Verbs (RDMA or Send/Recv), and MPI. In all these different solutions, gRPC is responsible for at least administrative tasks such as establishing the RDMA path, exchanging computation graphs, etc. Therefore, if gRPC is a compulsory component of TensorFlow, it makes more sense to bring RDMA capability directly into the gRPC runtime. This will allow TensorFlow to automatically benefit from RDMA-enhanced gRPC. In fact, there is an existing version of RDMA-based gRPC [13] in the community, which indicates researchers are investigating in this direction. With these many available channels, several important questions we need to explore: **1)** Can these new channels bring benefits for the DL workloads? **2)** Which channel performs the best and why? **3)** Is there any need to propose a new RDMA-based gRPC runtime, which can provide better performance than existing channels? **4)** If so, how much additional performance benefit can we gain through the proposed designs?

B. Contributions

To address above challenges, this paper first meticulously analyze the distributed TensorFlow communication characteristics to find the bottlenecks present in the existing supported channels. Through our characterization, we find many critical bottlenecks in all these existing channels (See Section III). We use these observations to guide us proposing a new gRPC runtime, called AR-gRPC. In AR-gRPC, we propose designs such as hybrid RDMA communication protocols, message pipelining and coalescing, zero-copy transmission etc. to make our runtime be adaptive to different message sizes for DL workloads (See Section IV). From our performance evaluations, we see that our proposed design can speedup gRPC performance by up to 4.1x and 2.3x compared to the default gRPC on IPoIB and the public RDMA-based gRPC, respectively. By integrating AR-gRPC with TensorFlow, we achieve up to 3x performance speedup for distributed training over default gRPC-IPoIB based TensorFlow (See Section V).

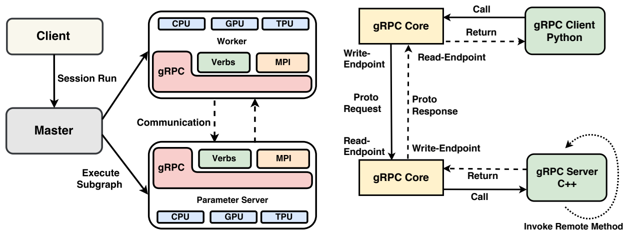
Through our proposed AR-gRPC design, TensorFlow can run with gRPC channel alone and get the optimal performance. We believe that our proposed design will significantly reduce the maintenance work for the TensorFlow community as well as simplify the usage for the end users. AR-gRPC can also benefit other applications or systems which are using gRPC.

II. Background

This section presents an overview on TensorFlow and gRPC.

A. Overview of TensorFlow

TensorFlow [12] is a widely adopted open source Deep Learning framework developed by the Google Brain Team. TensorFlow leverages data flow graphs to do the distributed deep neural network training. Nodes in the graph represent mathematical operations and the graph edges represent the multidimensional data arrays (i.e., tensors) communicated across the nodes. The execution model of distributed TensorFlow can be attributed to four distinct components: client, master, a set of workers, and several Parameter Servers (PS). Figure 1(a) illustrates the interaction among these components. The computational graph is built by a user-written client Tensorflow program. The client then creates a session to the master and sends the graph definitions as a protocol buffer. Afterwards, the master delegates and coordinates the execution (after pruning and optimizing) of the subgraphs to a set of distributed worker and PS processes. Each of these processes can use various devices (e.g., CPU, GPU, TPU [14]) to finish their task. The Parameter Servers are responsible for updating and storing the model parameters, while the workers send optimization updates of the model to and get the updated model from Parameter Servers. The parameter exchanging process (or tensor transmission) is the main communication phase, and the default open-source TensorFlow can support different communication channels such as gRPC, gRPC+Verbs, and gRPC+MPI to handle it, as shown in Figure 1(a).



(a) TensorFlow (b) gRPC

Figure 1: Overview of TensorFlow and gRPC

B. Overview of gRPC

gRPC [4] is the most fundamental communication mechanism in distributed TensorFlow. No matter which channel, as mentioned in Section II-A, is selected in TensorFlow, gRPC will be always used to perform communication for administrative tasks and (or) exchanging model parameters. Not only for TensorFlow, many other production systems in companies like Netflix, Cisco, Juniper etc. use gRPC for connecting multiple services in their environments. Figure 1(b) depicts a typical gRPC based communication scenario where a gRPC Python client communicates with a gRPC server that is written in C++. The client and server communicate with each other by using the protocol buffer protocol (i.e., Proto Request/Response). The gRPC core handles the communication and thus is the main point of interest. The gRPC core defines an abstraction called an Endpoint that encapsulates a channel between two communicating processes. An Endpoint implements Write and Read callbacks for a specific transport (for example TCP and UDP) protocol. In Section IV, we illustrate how we extend this design paradigm to support RDMA-based endpoints.

III. Characterization of Distributed TensorFlow

Before delving into designing new communication schemes, we first characterize the existing communication channels in TensorFlow to identify possible bottlenecks.

A. Methodology for Characterization

We characterize the three communication channels available in the open source TensorFlow. The default gRPC channel runs over IPoIB, while Verbs and MPI based channels use native RDMA based communication for tensor transmission.

We choose MVAPICH2-2.3b and Intel-MPI-2018 libraries for the MPI channel and found that both the MPI libraries provide similar results. We deploy a four-node TensorFlow cluster (i.e., Cluster A, see Section V-A) in the Parameter Server (PS) mode. The PS is deployed on one node (uses CPU), while the workers are deployed on the rest (use GPUs). We synchronously train (32/GPU batch size) a resnet50 [15] DNN, available in TensorFlow Convolution Neural Net benchmark [5]. This benchmark generates synthetic images and measures TensorFlow training performance by total number of images processed per second. Resnet50 DNN is a moderately complex network and thus is suitable for our analysis.

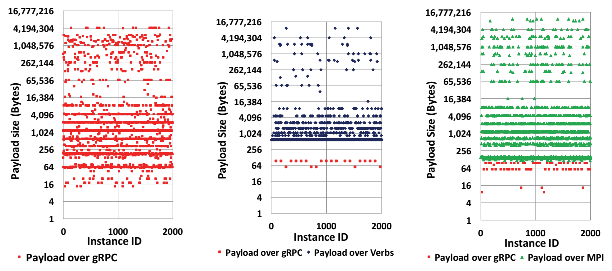
We have two important observations from the results summarized in Table II. First, gRPC+Verbs can perform slightly better than gRPC (i.e., 103.21 vs. 91.06), however, the benefit is not significant (around 13%). This implies that gRPC+Verbs can utilize the RDMA network efficiently compared to the default gRPC, but the question is can the performance be improved even further? Second, we surprisingly see that gRPC+MPI performs worse than the default gRPC. These two observations motivate us to further understand the in-depth designs and communication characteristics of TensorFlow with different channels. The following sections will present the characterization details.

Table II: TensorFlow Performance for Resnet50

Channel	Images/Sec
gRPC	91.06
gRPC+Verbs	103.21
gRPC+MPI	84.45

B. *Characterization for the gRPC Channel*

To understand the communication characteristics, we first profile the payload sizes being transmitted during TensorFlow training. Figure 2(a) shows 2K samples of payload distribution when the default gRPC channel is used. These snapshots are taken from one of the worker nodes as the other nodes have the similar traces due to the symmetrical characteristic of the workload. In Figure 2(a), we see that the communication over Socket-based gRPC channel involves a lot of short as well as large (up to 4 MBytes) messages. The reason of such upper bound is that the default gRPC has a maximum 4 MBytes payload limit. However, from later profiling results (see Figure 2(b) and 2(c)), we see that the actual payload in the training of resnet50 can be much larger than 4 MBytes. Clearly, such a naive chunking scheme in gRPC for transferring large messages with TCP/IP over a high-performance network is one of the major bottlenecks. To further identify potential bottlenecks, we analyze the communication flow for tensor transfer over gRPC channel, as shown in Figure 3(a). TensorFlow uses a rendezvous protocol for tensor transmission. The TF (TensorFlow) Sender always puts the tensors in the local table, whereas the TF receiver actively requests for the tensor only when needed. The default gRPC uses `sendmsg` and `recvmsg` primitives for sending and receiving payloads. These primitives are useful for sending or receiving from one or more buffers in a single function call. The payloads are constructed using Unix `iovec` structures. However, `sendmsg` internally copies all the data either into a pre-allocated (for payload less than 2KBytes), or a newly allocated buffer. This extra copying and allocation of new memory can be a bottleneck for high-speed data transfer.



(a) gRPC (b) gRPC+Verbs (c) gRPC+MPI

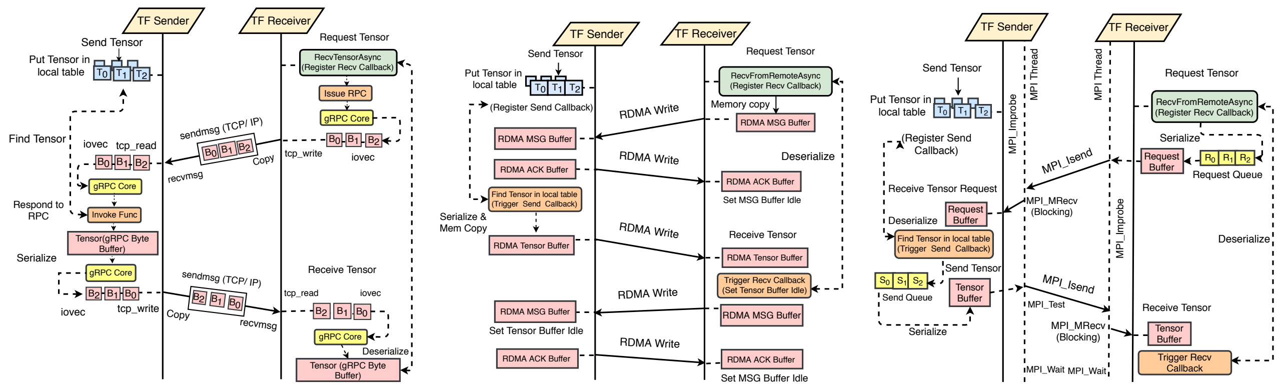
Figure 2: Payload distribution during Resnet50 training

C. *Characterization for the gRPC+Verbs Channel*

Similarly, Figure 2(b) shows the payload distributions over gRPC+Verbs. Here gRPC is responsible for administrative tasks and the tensor transfers are done over Verbs that is capable of RDMA. Figure 3(b) depicts the communication flow of tensor transfer over Verbs. The Verbs-based scheme writes all the payloads by employing an RDMA Write operation. Figure 2(b) shows that the Verbs-based channel is sending mostly 512 Bytes chunk payloads. However, studies [16, 17] have shown that writing messages of this length using RDMA rendezvous protocol is suboptimal. Also, using only RDMA Write for all payloads may not be the most efficient use of RDMA [17]. As shown in Figure 3(b), TF Sender and Receiver maintain two message buffers, two ACK buffers, and many tensor buffers. These buffers are pre-pinned RDMA buffers. For requesting a tensor, TF Receiver sends a message to notify the TF sender. TF Sender first sends an ACK so that TF Receiver can set the message buffer idle. Then TF sender finds the tensor locally and places at corresponding RDMA tensor buffer for transmission. When the tensor size increases, the current buffer is discarded and a new buffer of larger size is created and pinned. However, this process requires additional RDMA message exchanges between sender and receiver which is a bottleneck. We notice that for a single tensor transfer, several RDMA writes are involved for flow control and payload transmission. We aim to design better protocols to minimize the number of RDMA operations to further improve the performance.

D. *Characterization for the gRPC+MPI Channel*

In the gRPC+MPI channel, gRPC is still responsible for administrative operations, whereas the MPI channel (capable of RDMA) is used for transferring tensors. Figure 2(c) indicates a wide range of payloads starting from 128 Bytes to 10 MBytes over the MPI channel. In our experiments, the MPI channel needs a sufficient amount of tuning to get acceptable TensorFlow performance. Figure 3(c) shows the communication flow for tensor exchange via MPI. A dedicated MPI thread handles all the MPI calls in both sender and receiver side. The TF Receiver places the tensor requests in a Request Queue and the MPI-thread sends the requests to the



(a) gRPC (b) gRPC+Verbs (c) gRPC+MPI

Figure 3: Communication Flow of Tensor Transfer Over Different TensorFlow Channels

remote node using MPI_Isend. The remote node then forwards the request to the TensorFlow core that finds the tensor in the local table. Afterwards, a callback places the tensor in the Send Queue of the MPI-thread. The MPI-thread checks for a new tensor request or tensor using MPI_Improbe and uses MPI_MRecv (uses ANY_SOURCE) to receive data. Based on the studies [18, 19] in the HPC community, the default design of the MPI channel in TensorFlow has many bottlenecks. For example, the current communication flow heavily relies on the dedicated MPI thread which could be a bottleneck due to multi-threading, locking, and context switching overhead. The probing and blocking receiving with ANY_SOURCE also incur overhead due to internal wildcard matching logic in MPI. These are the reasons why we do not observe better performance in Table II for the MPI channel. Although, the message size distribution over gRPC remains almost similar for both Verbs and MPI channels, interestingly, we observe that the gRPC+MPI channel has more message transmissions over the gRPC. This is due to additional gRPC messages needed to set up different MPI ranks and assign tasks.

E. Summary

From these analysis, we have the following key observations: **1)** The training involves wide range of message transfers. Communication optimization for large tensors (e.g., 10 MBytes for resnet50) will reduce the training time, which is especially true if the DNN model is more complex. **2)** The default designs for the three channels in TensorFlow still have bottlenecks for utilizing RDMA-capable high-performance networks as discussed above. **3)** For both gRPC+Verbs and gRPC+MPI schemes, even though a small fraction, some messages still go over the default inefficient gRPC with TCP/IP. Also, both of these schemes still need to maintain two separate communication runtimes co-existing in the same TensorFlow architecture. This may cause inefficient communication performance due to resource contention, unawareness between each other, and possible deadlocks [20]. **4)** None of these channels support adaptive communication for Deep Learning workloads with different message sizes.

As we can see, a clear challenge is facing the TensorFlow community – **Can we propose a unified approach to have a single gRPC runtime in TensorFlow with adaptive and efficient RDMA protocols, which can resolve the bottlenecks as mentioned above?** Although there are some initial attempts in the community to integrate gRPC with RDMA, the design of existing version of RDMA-gRPC [13] is suboptimal due to several reasons, such as lack of using one-sided RDMA operations, no adaptive designs, interrupt-based signaling, etc. As we will see later in Section V, their design is not suitable for the transmission patterns that Deep Learning applications demand. Therefore, we propose a highly optimized adaptive gRPC with RDMA (i.e., AR-gRPC) that brings lower latency and higher throughput over high-speed interconnects.

IV. Proposed Design of AR-gRPC

In this section, we present AR-gRPC that brings high-performance RDMA-based communication over InfiniBand and RoCE. First, we discuss the key components of AR-gRPC architecture in Section IV-A. Then in Section IV-B. we discuss the associated optimizations for achieving high performance.

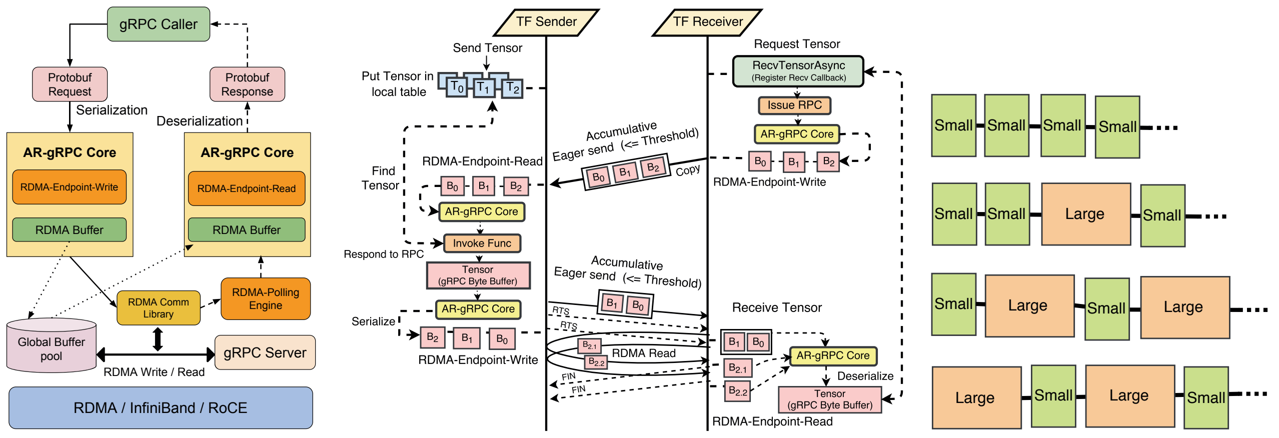
A. Architecture Overview of AR-gRPC

In AR-gRPC, we revamp the communication layer of the default gRPC architecture. We propose novel RDMA Endpoints that achieve low latency and high throughput. Figure 4(a) shows the architecture overview of AR-gRPC engine.

RDMA-Endpoint: RDMA-Endpoint extends the core communication abstraction (i.e., Endpoint) in default gRPC design, that encapsulates an RDMA connection between an RDMA client and server. This provides functionalities such as write (RDMA-Endpoint-Write), read (RDMA-Endpoint-Read), and polling (RDMA-Polling). RDMA-Endpoint can comply with the default gRPC Endpoint architecture seamlessly.

RDMA-Endpoint-Write: As shown in Figure 4(a), serialized protobuf messages from the application layer are sent to the remote process via RDMA-Endpoint-Write. RDMA-Endpoint-Write uses a pinned RDMA buffer for the message transfer.

RDMA-Polling: Due to the popularity of multi-core processors on modern clusters, we choose one or more dedicated cores to perform RDMA completion queue polling. Also, we employ "busy polling" completion detection strategy that is to repeatedly poll completion queue until a completion (sending or receiving a message) become available. In this way, the core resources are used efficiently and also aids in achieving low latency send/receive over the network. All new incoming



(a) Architecture of AR-gRPC (b) Communication Flow (c) gRPC `iovec` patterns for TensorFlow

Figure 4: Overview of AR-gRPC and the Corresponding Communication in TensorFlow

messages are kept on a global RDMA buffer pool. Once the polling engine detects a newly received message, it triggers RDMA-Endpoint-Read to consume the message.

RDMA-Endpoint-Read: Messages received by the RDMA-Polling thread are given to the RDMA-Endpoint-Read handler. The RDMA-Endpoint-Read handler then constructs an application payload from the contents in the RDMA pinned buffer. Afterwards, it sends the payload to the upper layer where it is subsequently deserialized and consumed by the application.

B. Adaptive RDMA-based Communication

As shown in Section III, DL workloads on TensorFlow involves many different size message transfers. All the existing designs in different TensorFlow channels are transferring messages in sub-optimal fashion. In this paper, we propose adaptive RDMA-based communication schemes as follows.

Hybrid Communication: We choose the eager protocol for small messages (with two-sided RDMA Send/Recv) and rendezvous (with one-sided RDMA READ from the remote side) protocol for the rest. The eager threshold of the message size is auto-tuned based on the underlying architecture. Moreover, this parameter is user tunable for added flexibility. Our proposed design uses RDMA operations more efficiently as compared to the Verbs-based channel we discussed in Section III. This is mainly because of three reasons: 1) Our design can adapt to the message sizes to automatically choose the proper protocol, while the communication protocol in the default Verbs-based channel of TensorFlow is fixed. 2) We choose RDMA Read as it sends only RTS message before the receiver reads from the remote memory. In contrast, the default Verbs-based channel in TensorFlow chooses to use RDMA Write based protocol, which needs to send multiple control (e.g., RTS and CTS) messages before writing to the remote memory. 3) Our design decouples the buffer management with RDMA communication that can give the best flexibility, while the default Verbs channel in TensorFlow has tightly coupled message buffers and ACK buffers for each RDMA connection [5]. Figure 4(b) represents the communication flow of tensor transfer when AR-gRPC is used in TensorFlow. As we can see from the figure, small payloads are transferred to the remote side using an eager protocol, while the large payload (especially when sending the requested tensor) is chunked and transferred using non-blocking one-sided RDMA READ in a pipelined fashion (as discussed below).

Message Pipelining and Coalescing: As discussed in Section III-B, due to different sizes of tensors and control messages, the gRPC channel needs to handle the underlying `iovec` buffers (asymmetric in size) properly to get the best communication performance. We run multiple DL models (as discussed in Section V) to identify the common patterns for these `iovec` buffers. Figure 4(c) shows the different patterns we observe during our experiments. A naive design choice for RDMA-Endpoint-Write is to copy all these `iovec` buffers into a pinned RDMA buffer and then issue a blocking send operation (i.e., wait until the completion). Even though this design can use RDMA, it suffers from a major drawback. If the application payload size is large, then RDMA-Endpoint-Write will block for a longer time, causing the achieved latency to be sub-optimal. To resolve this issue, we chunk large buffers into smaller sized payloads using a configurable threshold. The default chunk size is auto-tuned based on an architecture-aware algorithm. To achieve this, we maintain a pre-tuned parameter table for most of the recent architecture combinations including processor types, network interface speeds etc. Then, during runtime, our algorithm detects the node architecture and selects the best parameter value. After the payload is chunked, each of them is copied into the RDMA pinned buffer (we will discuss how to avoid this copy later). In order to achieve efficient pipelined transfers, we send out these pieces of messages to the remote process by leveraging a non-blocking rendezvous (RDMA Read by the remote process) protocol as shown in Figure 4(b). The multiple non-blocking sends can saturate the network bandwidth as much as possible, which suits for large-message transfers. In the receiver side, there is one major challenge for large message transfers - when the engine should trigger the RDMA-Endpoint-Read to consume the received chunks. One solution is to wait for all the chunks to arrive and then trigger RDMA-Endpoint-Read.

However, this solution hinders RDMA-Endpoint-Read to consume partially arrived messages, which means the RDMA-Endpoint-Read completely blocks until the entire message has arrived. To mitigate this issue, we devise a non-blocking RDMA-Endpoint-Read, which triggers receiving callbacks as soon as it receives a chunk of a message. Our design ensures that the order of a chunk in the message is preserved. This mechanism ensures high concurrency of receiving an entire large message. The above design is suitable for transmitting large `iovec` buffers. When gRPC application payloads have many small `iovec` buffers, sending these buffers individually would not be optimal. Instead, we coalesce them (up to eager send threshold), maintaining the relative order, into pinned RDMA buffer and send it using eager protocol. For small message transfers, eager protocol performs better than rendezvous [17], because it sends both control messages and payloads together. This increases the efficiency of our design.
Zero-Copy Transmission: Even though the above designs significantly improve the performance of gRPC, we notice that memory copying of large messages to the RDMA pinned buffer and vice-versa become bottlenecks. This extra copy is because between TensorFlow and gRPC, there is a layer to perform the serialization and deserialization. To achieve zero-copy transmission, we need to find a transparent approach to remove this copy. Through analyzing the TensorFlow design, we find that when TensorFlow sends a tensor, it first allocates a gRPC byte buffer which can be directly backed by an RDMA pinned buffer. In this way, we do not need to change any code in TensorFlow but just make small changes in gRPC runtime to pick up an RDMA pinned buffer from the buffer pool and then return it to TensorFlow. During tensor transmission, gRPC will directly serialize the tensor into the RDMA buffer without any copy. Similarly, RDMA-Endpoint-Read also leverages the zero-copy design by constructing the application payload directly from the RDMA pinned buffer instead of allocating new memory and copying the content.

V. Performance Evaluation

This section aims to answer the following questions: **(1)** What is the improvement in the performance of AR-gRPC compared to other gRPC designs? **(2)** How much benefit can TensorFlow extract by using AR-gRPC?

A. Experimental Setup

We use the following two clusters in our evaluation:
(1) Cluster A: Intel Broadwell Cluster (RI2-IB-EDR): We use up to twelve nodes on RI2 cluster. Each node is provisioned with Intel Broadwell (E5-2680-v4) dual fourteen-core processors, NVIDIA Tesla K80 GPU, 512 GB of memory, and a Mellanox IB EDR (100 Gbps) HCA. The host processors are running CentOS release 7.2.
(2) Cluster B: SDSC Comet (SDSC-Comet-IB-FDR): We use up to four nodes on this cluster. Each node is provisioned with Intel Haswell (E5-2680-v3) dual twelve-core processors, 128 GB of memory, and a Mellanox IB FDR (56 Gbps) HCA. The host processors are running CentOS release 6.7.

In all of our experiments, we use gRPC 1.5. AR-gRPC is also based on this version. The public RDMA-gRPC [13] is based on gRPC r0.14. gRPC is evaluated on both Cluster A and B, however, the public RDMA-gRPC fails to run on Cluster B as it hangs due to a race condition in their code. We use TensorFlow 1.3 in our experiments on Cluster A. We were unable to carry TensorFlow experiments on Cluster B due to GLIBC dependency issues. As Cluster B is public, we do not have the permission to install the required libraries.

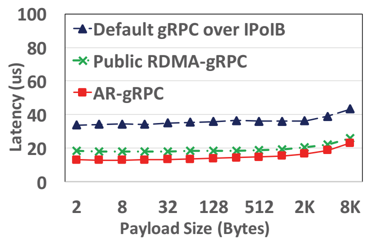
B. Evaluation of gRPC

We implement three RPC micro-benchmarks [21] to evaluate different gRPC designs. These benchmarks are - **(1)** Point-to-Point Latency, **(2)** Single Server, Multiple Clients Throughput, and **(3)** Performance Comparison in a Fully-Connected Architecture. We use gRPC C++ APIs to design these benchmarks. We run the benchmarks 1K times and report the average results. Note that in all the experiments the default Socket-based gRPC runs over IPoIB for a fair comparison.
Point-to-Point Latency: Figure 5 shows the comparison of Point-to-Point latency among different gRPC designs on Cluster A. Figure 6 shows the same comparison between default gRPC, and AR-gRPC on Cluster B. We categorize the payload sizes in three different classes - small, medium, and large ranging from 2 Bytes to 8 KBytes, 16 KBytes to 512 KBytes, and 1 MBytes to 8 MBytes, respectively. We choose these ranges because, from the characterization results in Section III, we see TensorFlow workloads contains all these message sizes.

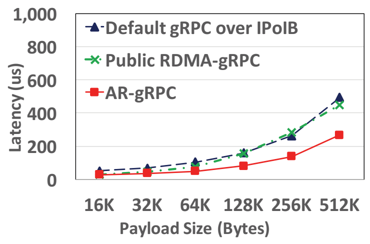
We first compare the results between the default gRPC and AR-gRPC. Figure 5(a) shows that the latency for 32 Bytes payload for default gRPC is 35.09 μs, whereas, AR-gRPC achieves 13.32 μs latency, resulting a 2.6x performance speedup. Also, Figure 5(b) and 5(c) show that AR-gRPC reduces the latency of 64 KBytes and 1 MBytes payload by 52% and 55% respectively. Figure 6(a) depicts that in Cluster B AR-gRPC reduces 32 Bytes latency by 60%. Figure 6(b) and 6(c) show a speedup of about 2.5x and 4.1x for 64 KBytes and 1 MBytes payload, respectively. This improvement over default gRPC is mainly attributed to the AR-gRPC's native RDMA design that can perform much better than IPoIB.

Similarly, Figure 5(a) and 5(b) show that AR-gRPC achieves 1.3x and 1.5x speedup for 32 Bytes and 64 KBytes payload, respectively, over the public RDMA-gRPC. As shown in Figure 5(c), Point-to-Point latency for 1 MBytes payload is 802.58 μs for public RDMA-gRPC, however, AR-gRPC incurs only 430.66 μs latency for the same payload. Thus, our design shows a significant speedup of about 1.8x. One key observation is that the performance of public RDMA-gRPC degrades significantly as the message size increases. The primary reason is public RDMA-gRPC uses IBV_WR_SEND (similar to our eager send for small messages) for transmitting payload of all sizes and does not have any advanced optimization.

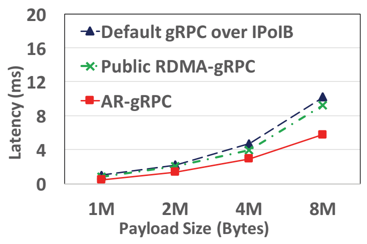
To further analyze the benefits of our design, Figure 7 depicts a latency comparison, using 512 KBytes to 8 MBytes payloads, among different AR-gRPC designs and public RDMA-gRPC. In this figure, the top line corresponds to public RDMA-gRPC and the rest depicts the incremental AR-gRPC



(a) Small Payload

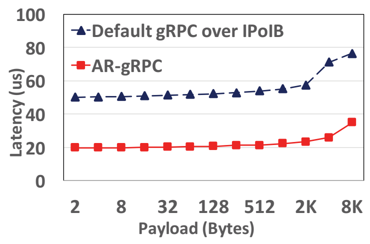


(b) Medium Payload

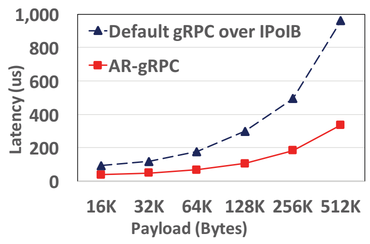


(c) Large Payload

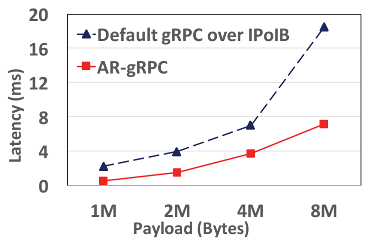
Figure 5: gRPC Point-to-Point Latency Evaluation on Cluster A



(a) Small Payload



(b) Medium Payload



(c) Large Payload

Figure 6: gRPC Point-to-Point Latency Evaluation on Cluster B

designs as discussed in Section IV-B. Public RDMA-gRPC performs worse even when we have only hybrid communication in our design. This proves that one-sided RDMA operation performs better in term of latency than two-sided Send-Receive for large messages. With incremental optimizations, we achieve even lower latency. The final version (bottom line in the figure) of AR-gRPC reduces latency of 8 MBytes message by 25% than the base AR-gRPC design (only has hybrid communication).

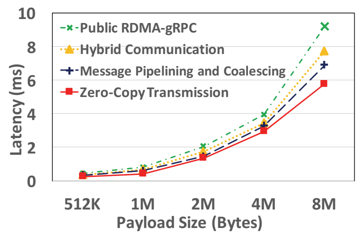


Figure 7: Analysis of Various gRPC Designs on Cluster A

Single Server, Multiple Clients Throughput: The throughput of an RPC system can be measured by the number of requests served per second by the server. Thus, this benchmark computes the total RPC requests handled by one server when multiple concurrent clients are sending requests. We use a fixed message size of 4KBytes. The server runs on one node, while, we vary the number of concurrent clients from 4 to 64 and distribute them uniformly among four different nodes.

Figure 8 represents the Single Server, Multiple Clients Throughput comparison among AR-gRPC, default gRPC and public RDMA-gRPC on Cluster A. In our experiment, we achieve at least 1.5x performance speedup in throughput compared to the default gRPC. For example, AR-gRPC achieves a throughput of 112,751 calls/s for 64 concurrent clients, whereas the default gRPC reaches 74,903 calls/s, resulting a 1.5x improvement. On the other hand, the public RDMA-gRPC fails to scale after 16 clients. For 64 clients, AR-gRPC design achieves a significant 2.3x speedup compared to public RDMA-gRPC design. We attribute this speedup to message pipelining and optimized RDMA-Polling as discussed in Section IV. A high throughput is desired for TensorFlow as, for example, in a large scale cluster all the workers may need to update variables in Parameter Server at once. Therefore, AR-gRPC is well-suited for that deployment.

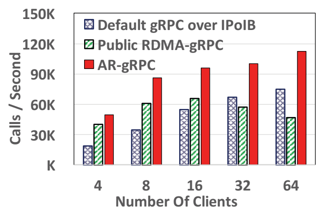
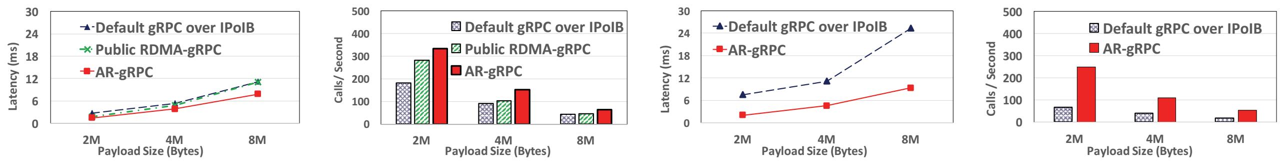


Figure 8: gRPC Single Server, Multiple Clients Throughput Evaluation on Cluster A

Performance Comparison in Fully-Connected Architecture: In a TensorFlow cluster, Node-to-Node communication is built by a gRPC server and multiple channels to connect with other workers' gRPC servers. This kind of deployment forms a Fully-Connected network architecture. For this experiment, the benchmark exactly models the communication pattern of TensorFlow. We deploy four nodes that spawn a gRPC server on each of the nodes and create three distinct gRPC channels that connect to other nodes' server. As in distributed TensorFlow, communication involves sending large tensors, we measure the performance by sending large payloads ranging from 2 MBytes to 8 MBytes. Figure 9 shows the performance comparison of different gRPC designs in terms of latency and throughput averaged across all the processes.

As shown in Figure 9(a), in Cluster A, AR-gRPC reduces average latency by 45% compared to the default gRPC for 2 MBytes payload. In addition, Figure 9(b) shows that AR-gRPC achieves about 1.8x and 1.18x average throughput speedup for 2 MBytes payload compared to default gRPC and public RDMA-gRPC, respectively. Also, Figure 9(c) and 9(d) show that in Cluster B, AR-gRPC achieves 60% reduction in average latency and obtains throughput speedup of about 2.68x for 4 Mbytes payload compared to default gRPC.



(a) Latency (A) (b) Throughput (A) (c) Latency (B) (d) Throughput (B)

Figure 9: Performance Comparison in Fully-Connected Architecture of gRPC

Results from the above experiments are a clear indication that compared with our proposed AR-gRPC design, the default gRPC on IPoIB or the public RDMA-gRPC do not achieve optimal performance over high-performance network. AR-gRPC outperforms default gRPC because the bottlenecks of Socket-based communication, memory copying etc., suppress the benefits of the high-performance network. In addition, as the public RDMA-gRPC implements RDMA sub-optimally, AR-gRPC's adaptive RDMA designs outperform significantly.

C. *Evaluation of AR-gRPC Enhanced TensorFlow*

In this section, we evaluate AR-gRPC enhanced TensorFlow with three other channels on Cluster A. We do not use the public RDMA-gRPC as it is incompatible with TensorFlow 1.3. In our experiments, we deploy TensorFlow in the Parameter Server (PS) mode on up to twelve nodes. One node hosts the PS and uses CPU, while the other nodes host the workers and use Nvidia Tesla K80 GPUs. We choose synchronous training over asynchronous as for the performance benefits [12, 22]. Also, we use different batch sizes for a comprehensive performance analysis. Note that the larger the batch size, the fewer the parameter updates, but also the higher the number of iterations needed for convergence. The maximum batch size we use is 32/GPU due to the GPU memory limit. 64/GPU batch size causes out of GPU memory error in TensorFlow is most of our experiments. We experiment with different DNNs from TensorFlow CNN benchmark [5]. This benchmark generates synthetic images and measures the performance by the total number of images processed. We run these tests five times and report the average result.

Inception4: Inception4 [23] is low computational cost DNN. Figure 10 shows the results on up to 12 nodes of Cluster A.

We observe that (Figure 10(a), 10(b), and 10(c)) AR-gRPC improves TensorFlow performance by a maximum of 29%, 80%, and 144% compared to default gRPC. For example, Figure 10(c) shows an improvement of 80% (93 vs 51 images) for batch size 16/GPU (total 176) on 12 nodes.

Moreover, in our experiments AR-gRPC process a maximum of 27%, 12%, and 31% more images than Verbs channel as shown in Figure 10(a), 10(b), and 10(c). Also, as shown in Figure 10(a), 10(b), and 10(c) AR-gRPC outperforms MPI channel by a maximum of 29%, 151%, and 228% for 4, 8, and 12 nodes, respectively. In our experiments, TensorFlow scales poorly with default MPI channel.

Resnet152: Resnet152 [15] is a popular residual DNN with a depth of 152 layers. Figure 11 represents the results of our experiments on up to twelve nodes of Cluster A.

Figure 11(c) shows that AR-gRPC incurs a maximum speedup of 3x (55 vs 18 images) compared to default gRPC. Even for higher batch size of 32/GPU (total 352) AR-gRPC improves TensorFlow performance by 82% (Figure 11(c)).

Figure 11(a), 11(b), and 11(c) show that AR-gRPC processes a maximum of 40%, 35%, and 30% more images, respectively, than Verbs. In addition, as seen in Figure 11(a). 11(b), and 11(c) AR-gRPC achieves a maximum speedup of 1.61x, 3.3x and 4.5x compared to MPI channel.

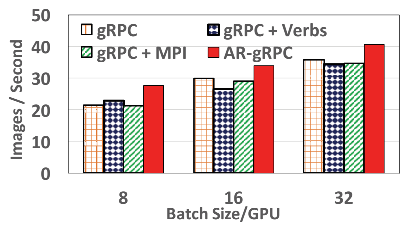
Inception3 and Resnet50: Figure 12(a) shows that for batch size of 16/GPU AR-gRPC improves Inception3 [24] performance by 53%, 26%, and 52% over default gRPC, Verbs, and MPI channels, respectively on 8 nodes of Cluster A. Figure 12(b) shows for Resnet50, AR-gRPC enhanced TensorFlow processes 47%, 20%, and 35% more images than gRPC-IPoIB, Verbs, and MPI channel for 32/GPU batch size.

GoogleNet and AlexNet: In this section, we compare results of two drastically different CNNs - GoogleNet [25] and AlexNet [26]. GoogleNet has only 5 Million parameters, whereas AlexNet has about 60 Million parameters. Figure 12(c) shows the comparison among gRPC and AR-gRPC on 8 nodes of Cluster A. We can not show other channels due to lack of space. AR-gRPC process a maximum of 128% (batch size 8/GPU) more images than default gRPC for GoogleNet. Although, for large batch size (32/GPU, total 224) the improvement is about 15% (597 vs 517). This is expected as higher batch size and less parameters in GoogleNet results in less network intensive gradient updates. In comparison, for the same batch size (32/GPU) AR-gRPC shows 89% (124 vs 65) performance improvement for Alexnet compared to default gRPC. This proves, even with higher batch size, if the DNN has a large number of parameters, AR-gRPC can improve TensorFlow performance significantly.

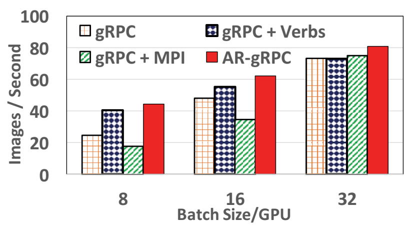
The above experiments show that AR-gRPC has the potential to accelerate Deep Learning using TensorFlow compared to the other available channels. Moreover, we show that when the DNN is complex and performs frequent network intensive variable updates, AR-gRPC provides the optimal channel over RDMA networks. Also, AR-gRPC scales well as compared to other channels with increasing number of nodes.

VI. Related Work

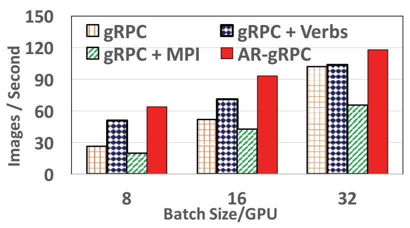
RPC over RDMA: Optimization of RPC is popular in the distributed computing field. The recent innovation of networking technologies powering large-scale data centers brings new challenges in terms of scaling and latency. Kalia et al. propose FaSST [27] RPC, to leverage the modern hardware. Stuedi et al. propose DaRPC [28] to implement tight integration between user space and RPC message passing. Moreover, SU et al. propose RDMA-based paradigm named RFP [29] that supports traditional RPC and provides high-performance. The



(a) 4 Nodes

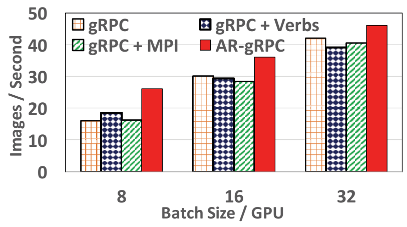


(b) 8 Nodes

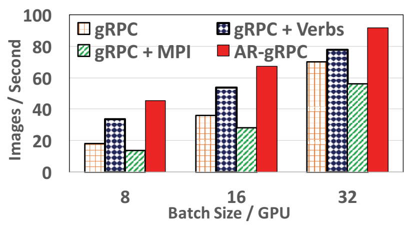


(c) 12 Nodes

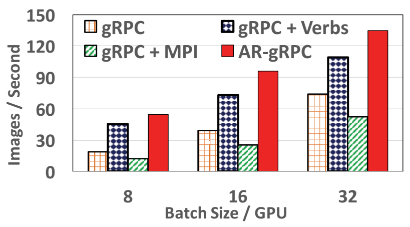
Figure 10: Inception4 Evaluation on Cluster A (Higher Better); $TotalBatchSize = (BatchSize/GPU) \times NUMofGPUs$



(a) 4 Nodes

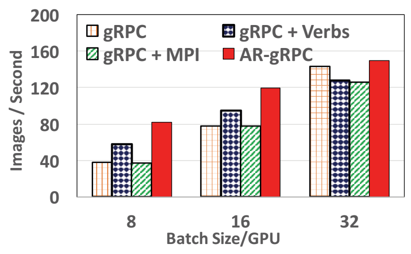


(b) 8 Nodes

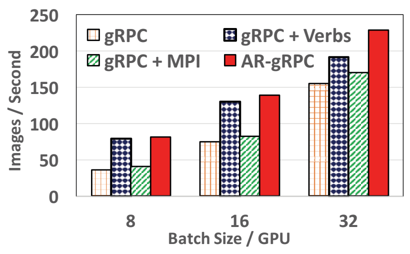


(c) 12 Nodes

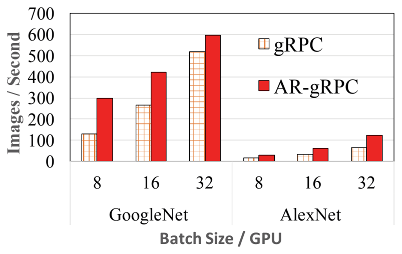
Figure 11: Resnet152 Evaluation on Cluster A (Higher Better); $TotalBatchSize = (BatchSize/GPU) \times NUMofGPUs$



(a) Inception3 (8 Nodes)



(b) Resnet50 (8 Nodes)



(c) GoogleNet & AlexNet (8 Nodes)

Figure 12: CNN Evaluation on Cluster A (Higher Better); $TotalBatchSize = (BatchSize/GPU) \times NUMofGPUs$

impact of a high-performance RPC is also well studied. For example, Lu et al. show high-performance Hadoop RPC [17] that benefits the entire Hadoop eco-system. In this paper, we have selected gRPC as it is a modern RPC framework that satisfies the need of the current data center requirements than other available open sourced RPCs. Even though there is a public RDMA-gRPC available, our design far exceeds that version in terms of performance.

Optimization of TensorFlow: Google's TensorFlow has been in the lime light for efficient Deep Learning in the recent time. Vishnu et al. extend TensorFlow [22] on large-scale cluster using MPI. By leveraging the high-performance optimized communication offered by MPI, they show good performance improvements. Jia [30] et al. propose RDMA TensorFlow similar to the official TensorFlow Verbs design. They report a 6x performance speedup over TCP/IP (1G Ethernet) gRPC, whereas AR-gRPC extended TensorFlow achieves 12x speedup over TCP/IP (1G Ethernet) gRPC. We don't show comparison numbers against TCP/IP over Ethernet in this paper to make a fair contrast. Lu [11] et al. have done a meticulous evaluation of popular Deep Learning frameworks over Big Data stacks on RDMA interconnects. They show that RDMA-based communication can lead to significant speed up in training time. TensorFlowOnSpark [31] is a framework proposed by Yahoo!, which allows execution of Deep Learning workloads with TensorFlow on existing Spark clusters.

Even though the primary and default communication of TensorFlow is powered by gRPC, TensorFlow's GitHub repository has contributions for supporting MPI and Verbs based channels. The primary reason for supporting these channels is that they can leverage high-performance communication mechanisms such as RDMA. However, in our paper, we present the potential in the direction of unification and adaptive designs. We argue that by making gRPC suitable for high-performance interconnects, we could achieve optimal performance for distributed TensorFlow training.

VII. Conclusion and Future Work

In this paper, we propose a high-performance adaptive RDMA-based communication runtime with gRPC (i.e., AR-gRPC) for distributed TensorFlow. The results suggest that by optimizing the performance of gRPC alone, we can achieve high-performance while keeping a unified communication runtime throughout the TensorFlow stack.This eliminates the need to maintain different server protocols for distributed TensorFlow. We perform a comprehensive analysis of TensorFlow architecture and propose an adaptive RDMA-enhanced gRPC runtime specifically designed for Deep Learning applications.We demonstrate that our AR-gRPC achieves 4.1x speedup compared to the default gRPC on IPoIB. We also show that AR-gRPC can speedup performance by 2.3x over the public RDMA-gRPC. Then, we show that AR-gRPC can benefit the runtime of distributed TensorFlow. We achieve

3x performance improvement when using AR-gRPC channel compared to using the default gRPC on IPoIB. Furthermore, AR-gRPC can benefit not only TensorFlow but other applications as well, such as micro-services running on modern data centers with gRPC as their common communication substrate.

We primarily focus on Parameter Server based TensorFlow training in this paper. As part of future work, we will explore different TensorFlow architectures including Uber's (horovod) and Baidu's reduce tree based collective communication design. We have made our design publicly available through RDMA-TensorFlow 0.9.1 package [32].

References

[1] S. Gupta, W. Zhang, and F. Wang, "Model Accuracy and Runtime Tradeoff in Distributed Deep Learning: A Systematic Study," in *International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 171–180.

[2] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware Async-SGD for Distributed Deep Learning," *arXiv preprint arXiv:1511.05950*, 2015.

[3] Y. Ren, X. Wu, L. Zhang, Y. Wang, W. Zhang, Z. Wang, M. Hack, and S. Jiang, "iRDMA: Efficient Use of RDMA in Distributed Deep Learning Systems," in *High Performance Computing and Communications*. IEEE, 2017, pp. 231–238.

[4] "gRPC - A High-Performance, Open-Source Universal RPC Framework." http://www.grpc.io/, 2018.

[5] "TensorFlow," 2018. [Online]. Available: https://github.com/tensorflow/

[6] MPI Forum, "MPI: A Message Passing Interface," in *Proceedings of Supercomputing*, 1993.

[7] X. Lu, M. W. U. Rahman, N. Islam, D. Shankar, and D. K. Panda, "Accelerating Spark with RDMA for Big Data Processing: Early Experiences," in *High-Performance Interconnects (HOTI), 2014 IEEE 22nd Annual Symposium*. IEEE, 2014, pp. 9–16.

[8] X. Lu, D. Shankar, S. Gugnani, and D. K. Panda, "High-Performance Design of Apache Spark with RDMA and Its Benefits on Various Workloads," in *2016 IEEE International Conference on Big Data (BigData '16)*. IEEE, 2016, pp. 253–262.

[9] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro, "FaRM: Fast Remote Memory," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, 2014, pp. 401–414.

[10] D. Shankar, X. Lu, N. Islam, M. Wasi-Ur-Rahman, and D. K. Panda, "High-Performance Hybrid Key-Value Store on Modern Clusters with RDMA Interconnects and SSDs: Non-blocking Extensions, Designs, and Benefits," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 393–402.

[11] X. Lu, H. Shi, M. H. Javed, R. Biswas, and D. K. Panda, "Characterizing Deep Learning over Big Data (DLoBD) Stacks on RDMA-Capable Networks," in *High-Performance Interconnects (HOTI), 2017 IEEE 25th Annual Symposium*. IEEE, 2017, pp. 87–94.

[12] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A System for Large-Scale Machine Learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[13] "Public RDMA Based gRPC," 2017. [Online]. Available: https://github.com/CGCL-codes/Tensorflow-RDMA/tree/master/src/grpc.git

[14] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning For Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[16] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High Performance RDMA-Based MPI Implementation over InfiniBand," in *Proceedings of the 17th Annual International Conference on Supercomputing*. ACM, 2003, pp. 295–304.

[17] X. Lu, D. Shankar, S. Gugnani, H. Subramoni, and D. K. Panda, "Impact of HPC Cloud Networking Technologies on Accelerating Hadoop RPC and HBase," in *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference*. IEEE, 2016, pp. 310–317.

[18] M. Li, X. Lu, S. Potluri, K. Hamidouche, J. Jose, K. Tomko, and D. K. Panda, "Scalable Graph500 Design with MPI-3 RMA," in *Cluster Computing (CLUSTER), 2014 IEEE International Conference*. IEEE, 2014, pp. 230–238.

[19] M. Li, X. Lu, K. Hamidouche, J. Zhang, and D. K. Panda, "Mizan-RMA: Accelerating Mizan Graph Processing Framework with MPI RMA," in *High Performance Computing (HiPC), 2016 IEEE 23rd International Conference*. IEEE, 2016, pp. 42–51.

[20] J. Jose, M. Luo, S. Sur, and D. K. Panda, "Unifying UPC and MPI Runtimes: Experience with MVAPICH," in *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model, PGAS 2010, New York, NY, USA, October 12-15, 2010*, 2010, p. 5.

[21] R. Biswas, X. Lu, and D. K. Panda, "Designing a Micro-Benchmark Suite to Evaluate gRPC for TensorFlow: Early Experiences," *arXiv preprint arXiv:1804.01138*, 2018.

[22] A. Vishnu, C. Siegel, and J. Daily, "Distributed Tensorflow with MPI," *arXiv preprint arXiv:1603.02339*, 2016.

[23] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, vol. 4, 2017, p. 12.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *CoRR*, vol. abs/1512.00567, 2015.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going Deeper with Convolutions." Cvpr, 2015.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[27] A. Kalia, M. Kaminsky, and D. G. Andersen, "FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs." in *OSDI*, 2016, pp. 185–201.

[28] P. Stuedi, A. Trivedi, B. Metzler, and J. Pfefferle, "DaRPC: Data Center RPC," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 15:1–15:13.

[29] M. Su, M. Zhang, K. Chen, Z. Guo, and Y. Wu, "RFP: When RPC is Faster than Server-Bypass with RDMA." in *EuroSys*, 2017, pp. 1–15.

[30] C. Jia, J. Liu, X. Jin, H. Lin, H. An, W. Han, Z. Wu, and M. Chi, "Improving the Performance of Distributed TensorFlow with RDMA," *International Journal of Parallel Programming*, pp. 1–12, 2017.

[31] "TesnorFlowOnSpark," 2018. [Online]. Available: https://github.com/yahoo/TensorFlowOnSpark

[32] "RDMA-TensorFlow 0.9.1," 2018. [Online]. Available: http://hidl.cse.ohio-state.edu/